MD-GAN: Multi-Discriminator Generative Adversarial Networks for Distributed Datasets

Corentin Hardy*, Erwan Le Merrer†, Bruno Sericola†
*Technicolor and Inria, †Univ Rennes, Inria, CNRS, IRISA
Corentin.Hardy@technicolor.com, erwan.le-merrer@inria.fr, bruno.sericola@inria.fr

***Abstract*—A recent technical breakthrough in the domain of machine learning is the discovery and the multiple applications of Generative Adversarial Networks (GANs). Those generative models are computationally demanding, as a GAN is composed of two deep neural networks, and because it trains on large datasets. A GAN is generally trained on a single server.**

In this paper, we address the problem of distributing GANs so that they are able to train over datasets that are spread on multiple workers. MD-GAN is exposed as the first solution for this problem: we propose a novel learning procedure for GANs so that they fit this distributed setup. We then compare the performance of MD-GAN to an adapted version of *federated learning* to GANs, using the MNIST, CIFAR10 and CelebA datasets. MD-GAN exhibits a reduction by a factor of two of the learning complexity on each worker node, while providing better or identical performances with the adaptation of federated learning. We finally discuss the practical implications of distributing GANs.

I. Introduction

Generative Adversarial Networks (GANs for short) are *generative* models, meaning that they are used to generate new realistic data from the probability distribution of the data in a given dataset. Those have been introduced by Goodfellow *et al* in seminal work [1]. Applications are for instance to generate pictures from text descriptions [2], to generate video from still images [3], to increase resolution of images [4], or to edit them [5]. Application to the chess game [6] or to anomaly detection [7] were also proposed, which highlights the growing and cross-domain interest from the machine learning research community towards GANs.

A GAN is a machine learning model, and more specifically a certain type of deep neural networks. As for all other deep neural networks, GANs require a large training dataset in order to fit the target application. Nowadays, the norm is then for service providers to collect large amounts of data (user data, application-specific data) into a central location such as their datacenter; the learning phase is taking place in those premises. The image super-resolution application [4] for instance leverages $350,000$ images from the ImageNet dataset; this application is representative of new advances: it provides state of the art results in its domain (measured in terms of quality of image reconstruction in that example); yet the question of computational efficiency or parallelism is left aside to future works.

The case was made recently for geo-distributed machine learning methods, where the data acquired at several datacenters stay in place [8], [9], as the considered data volumes would make it impossible to meet timing requirements in case of data centralization. Machine learning algorithms are thus to be adapted to that setup. Some recent works consider multiple generators and discriminators with the goal to improve GAN convergence [10], [11]; yet they do not aim at operating over spread datasets. The *Parameter Server* paradigm [12] is the prominent way of distributing the computation of classic (*i.e.*, non-GAN) neural networks: workers compute the neural network operations on their data share, and communicate the updates (gradients) to a central server named the parameter server. This framework is also the one leveraged for geo-distributed machine learning [8].

In this paper we propose MD-GAN, a novel method to train a GAN in a distributed fashion, that is to say over the data of a set of participating workers (*e.g.*, datacenters connected through WAN [8], or devices at the edge of the Internet [13]). GANs are specific in the sense that they are constituted of two different components: a *generator* and a *discriminator*. Both are tightly coupled, as they compete to reach the learning target. The challenges for an efficient distribution are numerous; first, that coupling requires fine grained distribution strategies between workers, so that the bandwidth implied by the learning process remains acceptable. Second, the computational load on the workers has to be reasonable, as the purpose of distribution is also to gain efficiency regarding the training on a single GPU setup for instance. Lastly, as deep learning computation has shown not to be a deterministic process when considering the accuracy of the learned models facing various distribution scales [14], the accuracy of the model computed in parallel has to remain competitive.

a) Contributions: The contributions of this paper are:
(i) to propose the first approach (MD-GAN) to distribute GANs over a set of worker machines. In order to provide an answer to the computational load challenge on workers, we remove half of their burden by having a single generator in the system, hosted by the parameter server. This is made possible by a peer-to-peer like communication pattern between the discriminators spread on the workers.
(ii) to compare the learning performance of MD-GAN with regards to both the baseline learning method (*i.e.*, on a standalone server) and an adaption of *federated learning* to GANs [15]. This permits head to head comparisons regarding the accuracy challenge.
(iii) to experiment MD-GAN and the two other competitors

on the MNIST, CIFAR10 and CelebA datasets, using GPUs. In addition to analytic expectations of communication and computing complexities, this sheds light on the advantages of MD-GAN, but also on the salient properties of the MD-GAN and federated learning approaches for the distribution of GANs.

b) Paper organization: In Section II, we give general background on GANs. Section III presents the computation setup we consider, and presents an adaptation of federated learning to GANs. Section IV details the MD-GAN algorithm. We experiment MD-GAN and its competitors in Section V. In Section VI, we review the related work. We finally discuss future works and conclude in Section VII.

II. Background on Generative Adversarial Networks

The particularity of GANs as initially presented in [1] is that their training phase is *unsupervised*, *i.e.*, no description labels are required to learn from the data. A classic GAN is composed of two elements: a *generator* $\mathcal{G}$ and a *discriminator* $\mathcal{D}$. Both are deep neural networks (DNN). The generator takes as input a noise signal (*e.g.*, random vectors of size k where each entry follows a normal distribution $\mathcal{N}(0,1)$) and generates data with the same format as training dataset data (*e.g.*, a picture of 128x128 pixels and 3 color channels). The discriminator receives as input either some data from two sources: from the generator or from the training dataset. The goal of the discriminator is to guess from which source the data is coming from. At the beginning of the learning phase, the generator generates data from a probability distribution and the discriminator quickly learns how to differentiate that generated data from the training data. After some iterations, the generator learns to generate data which are closer to the dataset distribution. If it eventually turns out that the discriminator is not able to differentiate both, this means that the generator has learned the distribution of the data in the training dataset (and thus has learned an unlabeled dataset in an unsupervised way).

Formally, let a given training dataset be included in the data space X, where $\boldsymbol{x}$ in that dataset follows a distribution probability P_{data}. A GAN, composed of generator $\mathcal{G}$ and discriminator $\mathcal{D}$, tries to learn this distribution. As proposed in the original GAN paper [1], we model the generator by the function $\mathcal{G}_{\boldsymbol{w}} : \mathbb{R}^{\ell} \longrightarrow X$, where $\boldsymbol{w}$ contains the parameters of its DNN $\mathcal{G}_{\boldsymbol{w}}$ and ℓ is fixed. Similarly, we model the discriminator by the function $\mathcal{D}_{\boldsymbol{\theta}} : X \longrightarrow [0,1]$ where $\mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the probability that $\boldsymbol{x}$ is a data from the training dataset, and $\boldsymbol{\theta}$ contains the parameters of the discriminator $\mathcal{D}_{\boldsymbol{\theta}}$. Writing $\log$ for the logarithm to the base 2, the learning consists in finding the parameters $\boldsymbol{w}^*$ for the generator:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \max_{\boldsymbol{\theta}} (A_{\boldsymbol{\theta}} + B_{\boldsymbol{\theta},\boldsymbol{w}}), \text{ with}$$

$$A_{\boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}} \left[\log \mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \text{ and}$$

$$B_{\boldsymbol{\theta},\boldsymbol{w}} = \mathbb{E}_{\boldsymbol{z}\sim \boldsymbol{\mathcal{N}}_{\ell}} \left[\log\left(1 - \mathcal{D}_{\boldsymbol{\theta}}\left(\mathcal{G}_{\boldsymbol{w}}(\boldsymbol{z})\right)\right)\right],$$

where $\boldsymbol{z} \sim \boldsymbol{\mathcal{N}}_{\ell}$ means that each entry of the ℓ-dimensional random vector $\boldsymbol{z}$ follows a normal distribution with fixed parameters. In this equation, $\mathcal{D}$ adjusts its parameters $\boldsymbol{\theta}$ to maximize $A_{\boldsymbol{\theta}}$, *i.e.*, the expected good classification on real data and $B_{\boldsymbol{\theta},\boldsymbol{w}}$, the expected good classification on generated data. $\mathcal{G}$ adjusts its parameters $\boldsymbol{w}$ to minimize $B_{\boldsymbol{\theta},\boldsymbol{w}}$ ($\boldsymbol{w}$ does not have impact on A), which means that it tries to minimize the expected good classification of $\mathcal{D}$ on generated data. The learning is performed by iterating two steps, named the discriminator learning step and the generator learning step, as described in the following.

1) Discriminator learning: The first step consists in learning $\boldsymbol{\theta}$ given a fixed $\mathcal{G}_{\boldsymbol{w}}$. The goal is to approximate the parameters $\boldsymbol{\theta}$ which maximize $A_{\boldsymbol{\theta}} + B_{\boldsymbol{\theta},\boldsymbol{w}}$ with the actual $\boldsymbol{w}$. This step is performed by a gradient descent (generally using the Adam optimizer [16]) of the following discriminator error function J_{disc} on parameters $\boldsymbol{\theta}$:

$$J_{disc}(X_r, X_g) = \widetilde{A}(X_r) + \widetilde{B}(X_g), \text{ with}$$

$$\widetilde{A}(X_r) = \frac{1}{b}\sum_{\boldsymbol{x}\in X_r} \log(\mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x})); \widetilde{B}(X_g) = \frac{1}{b}\sum_{\boldsymbol{x}\in X_g} \log(1-\mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x})),$$

where X_r is a batch of b real data drawn randomly from the training dataset and X_g a batch of b generated data from $\mathcal{G}$. In the original paper [1], the authors propose to perform few gradient descent iterations to find a good $\boldsymbol{\theta}$ against the fixed $\mathcal{G}_{\boldsymbol{w}}$.

2) Generator learning: The second step consists in adapting $\boldsymbol{w}$ to the new parameters $\boldsymbol{\theta}$. As done for step 1), it is performed by a gradient descent of the following error function J_{gen} on generator parameters $\boldsymbol{w}$:

$$\begin{aligned} J_{gen}(Z_g) &= \widetilde{B}\left(\{\mathcal{G}_{\boldsymbol{w}}(\boldsymbol{z}) | \boldsymbol{z} \in Z_g\}\right) \\ &= \frac{1}{b} \sum_{\boldsymbol{x}\in\{\mathcal{G}_{\boldsymbol{w}}(\boldsymbol{z})|\boldsymbol{z}\in Z_g\}} \log(1 - \mathcal{D}_{\boldsymbol{\theta}}(\boldsymbol{x})) \\ &= \frac{1}{b} \sum_{\boldsymbol{z}\in Z_g} \log(1 - \mathcal{D}_{\boldsymbol{\theta}}(\mathcal{G}_{\boldsymbol{w}}(\boldsymbol{z}))) \end{aligned}$$

where Z_g is a sample of b $\ell-$dimensional random vectors generated from $\boldsymbol{\mathcal{N}}_{\ell}$. Contrary to discriminator learning step, this step is performed only once per iteration.

By iterating those two steps a significant amount of times with different batches (see *e.g.*, [1] for convergence relaessentiallyted questions), the GAN ends up with a $\boldsymbol{w}$ which approximates $\boldsymbol{w}^*$ well. Such as for standard deep learning, guarantees of convergence are weak [17]. Despite this very recent breakthrough, there are lots of alternative proposals to learn a GAN (*e.g.*, more details can be found in [18], [19], and [20]).

III. Distributed computation setup for GANs

Before we present MD-GAN in the next Section, we introduce the distributed computation setup considered in this paper, and an adaptation of federated learning to GANs.

a) Learning over a spread dataset: We consider the following setup. N workers (possibly from several datacenters [8]) are each equipped with a local dataset composed of m samples (each of size d) from the same probability distribution P_{data} (*e.g.*, requests to a voice assistant, holiday pictures). Those local datasets will remain in place (*i.e.*, will not be sent over the network). We denote by $\mathcal{B} = \bigcup_{n=1}^{N} \mathcal{B}_n$ the entire dataset, with $\mathcal{B}_n$ the dataset local to worker n. We assume in the remaining of the paper that the local datasets are $i.i.d.$ on workers, that is to say that there are no bias in the distribution of the data on one particular worker node.

The assumption on the fix location of data shares is complemented by the use of the parameter server framework we are now presenting.

b) The parameter server framework: Despite the general progress of distributed computing towards serverless operation even in datacenters (*e.g.*, use of the *gossip* paradigm as in Dynamo [21] back in 2007), the case of deep learning systems is specific. Indeed, the amounts of data required to train a deep learning model, and the very iterative nature of the learning tasks (learning on batches of data, followed by operations of back-propagations) makes it necessary to operate in a parallel setup, with the use of a central server. Introduced by Google in 2012 [22], the *parameter server* framework uses *workers* for parallel processing, while one or a few central servers are managing shared states modified by those workers (for simplicity, in the remaining of the paper, we will assume the presence of a single central server). The method aims at training the same model on all workers using their given data share, and to synchronize their learning results with the server at each iteration, so that this server can update the model parameters.

Note that more distributed approaches for deep learning, such as gossip-based computation [23], [24], have not yet proven to work efficiently on the data scale required for modern applications; we thus leverage a variant of parameter server framework as our computation setup.

c) FL-GAN: adaptation of federated learning to GANs: By the design of GANs, a generator and a discriminator are two separate elements that are yet tightly coupled; this fact makes it nevertheless possible to consider adapting a known computation method, that is generally used for training a single deep neural network.[1] Federated learning [27] proposes to train a machine learning model, and in particular a deep neural network, on a set of workers. It follows the parameter server framework, with the particularity that workers perform numerous local iterations between each communication to the server (*i.e.*, a round), instead of sending small updates. All workers are not necessarily active at each round; to reduce conflicting updates, all active workers synchronize their model with the server at the beginning of each round.

[1]We note that more advanced GAN techniques such as those by Wang et al. [25] or by Tolstikhin et al. [26] might also be distributed and serve as baselines; yet this distribution requires a full redesign of the proposed protocols, and is thus out of the scope of this paper.

In order to compare MD-GAN to a federated learning type of setup, we propose an adapted version of federated learning to GANs. This adaptation considers the discriminator $\mathcal{D}$ and generator $\mathcal{G}$ on each worker as one computational object to be treated atomically. Workers perform iterations locally on their data and every E *epochs* (*i.e.*, each worker passes E times the data in their GAN) they send the resulting parameters to the server. The server in turn averages the $\mathcal{G}$ and $\mathcal{D}$ parameters of all workers, in order to send updates to those workers at the next iteration. We name this adapted version FL-GAN; it is depicted by Figure 1 b).

We now detail MD-GAN, our proposal for the learning of GANs over workers and their local datasets.

IV. The MD-GAN algorithm

A. Design rationale

To diminish computation on the workers, we propose to operate with a single $\mathcal{G}$, hosted on the server[2]. That server holds parameters $\boldsymbol{w}$ for $\mathcal{G}$; data shares are split over workers. To remove part of the burden from the server, discriminators are solely hosted by workers, and move in a peer-to-peer fashion between them. Each worker n starts with its own discriminator $\mathcal{D}_n$ with parameters $\boldsymbol{\theta}_n$. Note that the architecture and initial parameters of $\mathcal{D}_n$ could be different on every worker n; for simplicity, we assume that they are the same. This architecture is presented on Figure 1 a).

The goal for GANs is to train generator $\mathcal{G}$ using $\mathcal{B}$. In MD-GAN, the $\mathcal{G}$ on the server is trained using the workers and their local shares. It is a 1-versus-N game where $\mathcal{G}$ faces all $\mathcal{D}_n$, *i.e.*, $\mathcal{G}$ tries to generate data considered as real by all workers. Workers use their local datasets $\mathcal{B}_n$ to differentiate generated data from real data. Training a generator is an iterative process; in MD-GAN, a *global learning iteration* is composed of four steps:

- The server generates a set K of κ batches $K = \{X^{(1)}, \ldots, X^{(\kappa)}\}$, with $\kappa \leq N$. Each $X^{(i)}$ is composed of b data generated by $\mathcal{G}$. The server then selects, for each worker n, two distinct batches, say $X^{(i)}$ and $X^{(j)}$, which are sent to worker n and locally renamed as $X_n^{(g)}$ and $X_n^{(d)}$. The way in which the two distinct batches are selected is discussed in Section IV-B1.
- Each worker n performs L learning iterations on its discriminator $\mathcal{D}_n$ (see Section II-1) using $X_n^{(d)}$ and $X_n^{(r)}$, where $X_n^{(r)}$ is a batch of real data extracted locally from $\mathcal{B}_n$.
- Each worker n computes an error feedback F_n on $X_n^{(g)}$ by using $\mathcal{D}_n$ and sends this error to the server. We detail in Section IV-B2 the computation of F_n.
- The server computes the gradient of J_{gen} for its parameters $\boldsymbol{w}$ using all the F_n feedbacks. It then updates

[2]In that regard, MD-GAN do not fully comply with the parameter server model, as the workers do not compute and synchronize to the same model architecture hosted at the server. Yet, it leverages the parallel computation and the iterative nature of the learning task proposed by the the parameter server framework.

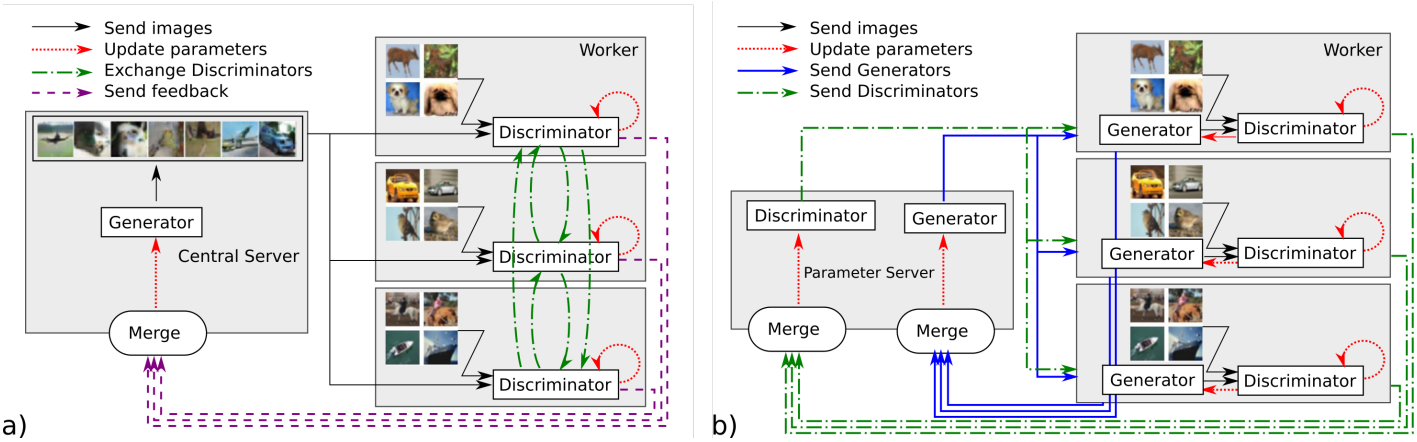

Figure 1: The two proposed competitors for the distribution of GANs: a) The MD-GAN communication pattern, compared to b) FL-GAN (federated learning adapted to GANs). MD-GAN leverages a single generator, placed on the server; FL-GAN uses generators on the server and on each worker. MD-GAN swaps discriminators between workers in a peer-to-peer fashion, while in FL-GAN they remain fix and are averaged by the server upon reception from the workers.

Notation	
$\mathcal{G}$	Generator
$\mathcal{D}$	Discriminator
N	Number of workers
C	Central server
W_n	Worker n
P_{data}	Data distribution
$P_{\mathcal{G}}$	Distribution of generator $\mathcal{G}$
$\boldsymbol{w}$ (resp. $\boldsymbol{\theta}$)	Parameters of $\mathcal{G}$ (resp. $\mathcal{D}$)
w_k (resp. θ_k)	k-th parameter of $\mathcal{G}$ (resp. $\mathcal{D}$)
$\mathcal{B}$	Distributed training dataset
$\mathcal{B}_n$	Local training dataset on worker n
m	Number of objects in a local dataset $\mathcal{B}_n$
d	Object size (*e.g.*, image in Mb)
b	Batch size
I	Number of training iterations
K	The set of all batches $X^{(1)}, \ldots, X^{(\kappa)}$ generated by $\mathcal{G}$ during one iteration
F_n	The error feedback computed by worker n
E	Number of local epochs before swapping discriminators

Table I: Table of notations

its parameters with the chosen optimizer algorithm (*e.g.*, Adam [16]).

Moreover, every E epochs, workers start a peer-to-peer swapping process for their discriminators, using function SWAP(). The pseudo-code of MD-GAN, including those steps, is presented in Algorithm 1.

Note that extra workers can enter the learning task if they enter with a pre-trained discriminator (*e.g.*, a copy of another worker discriminator); we discuss worker failures in Section V.

B. The generator learning procedure (server-side)

The server hosts generator $\mathcal{G}$ with its associated parameters $\boldsymbol{w}$. Without loss of generality, this paper exposes the training of GANs for image generation; the server generates new images to train all discriminators and updates $\boldsymbol{w}$ using error feedbacks.

1) Distribution of generated batches: Every global iteration, $\mathcal{G}$ generates a set of k batches $K = \{X^{(1)}, \ldots, X^{(\kappa)}\}$ (with $\kappa \leq N$) of size b. Each participating worker n is sent two batches among K, $X_n^{(g)}$ and $X_n^{(d)}$. This two-batch generation design is required, for the computation of the gradients for both $\mathcal{D}$ and $\mathcal{G}$ on separate data (such as for the original GAN design [1]). A possible way to distribute the $X^{(i)}$ among the N workers could be to set $X_n^{(g)} = X^{((n \mod k)+1)}$ and $X_n^{(d)} = X^{(((n+1) \mod \kappa)+1)}$ for $n = 1, \ldots, N$.

2) Update of generator parameters: Every global iteration, the server receives the error feedback F_n from every worker n, corresponding to the error made by $\mathcal{G}$ on $X_n^{(g)}$. More formally, F_n is composed of b vectors $\{\boldsymbol{e}_{n_1}, \ldots, \boldsymbol{e}_{n_b}\}$, where $\boldsymbol{e}_{n_i}$ is given by

$$\boldsymbol{e}_{n_i} = \frac{\partial \widetilde{B}(X_n^{(g)})}{\partial \boldsymbol{x}_i},$$

with $\boldsymbol{x}_i$ the i-th data of batch $X_n^{(g)}$. The gradient $\Delta \boldsymbol{w} = \partial \widetilde{B}\left(\cup_{n=1}^{N} X_n^{(g)}\right)/\partial \boldsymbol{w}$ is deduced from all F_n as

$$\Delta w_k = \frac{1}{Nb} \sum_{n=1}^{N} \sum_{\boldsymbol{x}_i \in X_n^{(g)}} \boldsymbol{e}_{n_i} \frac{\partial \boldsymbol{x}_i}{\partial w_k},$$

with Δw_k the k-th element of $\Delta \boldsymbol{w}$. The term $\partial \boldsymbol{x}_i / \partial w_k$ is computed on the server. Note that $\cup_{n=1}^{N} X_n^{(g)} = \{\mathcal{G}_w(\boldsymbol{z}) | \boldsymbol{z} \in Z_g\}$. Minimizing $\widetilde{B}\left(\cup_{n=1}^{N} X_n^{(g)}\right)$ is thus equivalent to minimize $J_{gen}(Z_g)$. Once the gradients are computed, the server is able to update its parameters $\boldsymbol{w}$. We thus choose to merge the feedback updates through an averaging operation, as it is the most common way to aggregate updates processed in parallel [28], [22], [29], [30]. Using the Adam optimizer [16], parameter $w_k \in \boldsymbol{w}$ at iteration t, denoted by $w_k(t)$ here, is computed as follows:

$$w_k(t) = w_k(t-1) + \text{Adam}(\Delta w_k),$$

Algorithm 1 The MD-GAN algorithm

procedure WORKER($C, \mathcal{B}_n, I, L, b$)
  Initialize $\boldsymbol{\theta}_n$ for $\mathcal{D}_n$
  for $i \leftarrow 1$ **to** I **do**
    $X_n^{(r)} \leftarrow$ SAMPLES($\mathcal{B}_n, b$)
    $X_n^{(g)}, X_n^{(d)} \leftarrow$ RECEIVEBATCHES(C)
    for $l \leftarrow 0$ **to** L **do**
      $\mathcal{D}_n \leftarrow$DISCLEARNINGSTEP(J_{disc},$\mathcal{D}_n$)
    end for
    $F_n \leftarrow \{\frac{\partial \widetilde{B}(X_n^{(g)})}{\partial \boldsymbol{x}_i} | \boldsymbol{x}_i \in X_n^{(g)}\}$
    SEND(C, F_n) ▷ Send F_n to server
    if $i \mod (\frac{mE}{b}) = 0$ **then**
      $\mathcal{D}_n \leftarrow$SWAP($\mathcal{D}_n$)
    end if
  end for
end procedure

procedure SWAP($\mathcal{D}_n$)
  $W_l \leftarrow$ GETRANDOMWORKER()
  SEND($W_l, \mathcal{D}_n$) ▷ Send $\mathcal{D}_n$ to worker W_l.
  $\mathcal{D}_n \leftarrow$ RECEIVED() ▷ Receive a new discriminator from another worker.
  Return $\mathcal{D}_n$
end procedure

procedure SERVER(κ,I) ▷ Server C
  Initialize $\boldsymbol{w}$ for $\mathcal{G}$
  for $i \leftarrow 1$ **to** I **do**
    for $j \leftarrow 0$ **to** κ **do**
      $Z_j \leftarrow$GAUSSIANNOISE(b)
      $X^{(j)} \leftarrow \{\mathcal{G}_{\boldsymbol{w}}(\boldsymbol{z}) | \boldsymbol{z} \in Z_j\}$
    end for
    $X_1^{(d)}, \ldots, X_n^{(d)} \leftarrow$ SPLIT($X^{(1)}, \ldots, X^{(\kappa)}$)
    $X_1^{(g)}, \ldots, X_n^{(g)} \leftarrow$ SPLIT($X^{(1)}, \ldots, X^{(\kappa)}$)
    for $n \leftarrow 1$ **to** N **do**
      SEND($W_n, (X_n^{(d)}, X_n^{(g)})$)
    end for
    $F_1, \ldots, F_N \leftarrow$ GETFEEDBACKFROMWORKERS()
    Compute $\Delta \boldsymbol{w}$ according to $F_1, \ldots, F_N$
    for $w_k \in \boldsymbol{w}$ **do**
      $w_k \leftarrow w_k +$ADAM(Δw_k)
    end for
  end for
end procedure

where the Adam optimizer is the function which computes the update given the gradient Δw_k.

3) Workload at the server: Placing the generator on the server increases its workload. It generates κ batches of b data using $\mathcal{G}$ during the first step of a global iteration, and then receives N error feedbacks of size bd in the third step. The batch generation requires $\kappa b G_{op}$ floating point operations (where G_{op} is the number of floating operations to generate one data object with $\mathcal{G}$) and a memory of $\kappa b G_a$ (with G_a the number of neurons in $\mathcal{G}$). For simplicity, we assume that $G_{op} = O(|\boldsymbol{w}|)$ and that $G_a = O(|\boldsymbol{w}|)$. Consequently the batch generation complexity is $O(\kappa b|\boldsymbol{w}|)$. The merge operation of all feedbacks F_n and the gradient computations imply a memory and computational complexity of $O(b(dN + \kappa|\boldsymbol{w}|))$.

4) The complexity vs. data diversity trade-off: At each global iteration, the server generates κ batches, with $\kappa < N$. If $\kappa = 1$, all workers receive and compute their feedback on the same training batch. This reduces the diversity of feedbacks received by the generator but also reduces the server workload. If $\kappa = N$, each worker receives a different batch, thus no feedback has conflict on some concurrently processsed data. In consequence, there is a trade-off regarding the generator workload: because $\kappa = N$ seems cumbersome, we choose $\kappa = 1$ or $\kappa = \lfloor \log(N) \rfloor$ for the experiments, and assess the impact of those values on final model performances.

C. The learning procedure of discriminators (worker-side)

Each worker n hosts a discriminator $\mathcal{D}_n$ and a training dataset $\mathcal{B}_n$. It receives batches of generated images split in two parts: $X_n^{(d)}$ and $X_n^{(g)}$. The generated images $X_n^{(d)}$ are used for training $\mathcal{D}_n$ to discriminate those generated images from real images. The learning is performed as a classical deep learning operation on a standlone server [1]. A worker n computes the gradient $\Delta \boldsymbol{\theta}_n$ of the error function J_{disc} applied to the batch of generated images $X_n^{(d)}$, and a batch or real image $X_n^{(r)}$ taken from $\mathcal{B}_n$. As indicated in Section II-1, this operation is iterated L times. The second batch $X_n^{(g)}$ of generated images is used to compute the error term F_n of generator $\mathcal{G}$. Once computed, F_n is sent to the server for the computation of gradients $\Delta \boldsymbol{w}$.

1) The swapping of discriminators: Each discriminator n solely uses $\mathcal{B}_n$ to train its parameters $\boldsymbol{\theta}_n$. If too many iterations are performed on the same local dataset, the discriminator tends to over specialize (which decreases its capacity of generalization). This effect, called *overfitting*, is avoided in MD-GAN by swapping the parameters of discriminators $\boldsymbol{\theta}_n$ between workers after E epochs. The swap is implemented in a gossip fashion, by choosing randomly for every worker another worker to send its parameters to.

2) Workload at workers: The goal of MD-GAN is to reduce the workload of workers without moving data shares out of their initial location. Compared to our proposed adapted federated learning method FL-GAN, the generator task is deported on the server. Workers only have to handle their discriminator parameters θ_n and to compute error feedbacks after L local iterations. Every global iteration, a worker performs $2bD_{op}$ floating point operations (where D_{op} is the number of floating point operations for a feed-forward step of $\mathcal{D}$ for one data object). The memory used at a worker is $O(|\boldsymbol{\theta}|)$.

D. The characteristic complexities of MD-GAN

1) Communication complexity: In the MD-GAN algorithm there are three types of communications:

- Server to worker communication: the server sends its κ batches of generated images to workers at the beginning of global iterations. The number of generated images is

	FL-GAN	MD-GAN
Computation C	$O(IbN(\|\boldsymbol{w}\|+\|\boldsymbol{\theta}\|)/(mE))$	$O(Ib(dN+\kappa\|\boldsymbol{w}\|))$
Memory C	$O(N(\|\boldsymbol{w}\|+\|\boldsymbol{\theta}\|))$	$O(b(dN+\kappa\|\boldsymbol{w}\|)$
Computation W	$O(Ib(\|\boldsymbol{w}\|+\|\boldsymbol{\theta}\|))$	$O(Ib\|\boldsymbol{\theta}\|)$
Memory W	$O(\|\boldsymbol{w}\|+\|\boldsymbol{\theta}\|)$	$O(\|\boldsymbol{\theta}\|)$

Table II: Computation and memory complexities for MD-GAN and FL-GAN. The rows in grey highlight the reduction by a factor of two for MD-GAN on workers.

Communication type	FL-GAN	MD-GAN
C→W (C)	$N(\boldsymbol{\theta}+\boldsymbol{w})$	bdN
C→W (W)	$\boldsymbol{\theta}+\boldsymbol{w}$	bd
W→C (W)	$\boldsymbol{\theta}+\boldsymbol{w}$	bd
W→C (C)	$N(\boldsymbol{\theta}+\boldsymbol{w})$	bdN
Total # C↔D	$Ib/(mE)$	I
W→W (W)	-	$\boldsymbol{\theta}$
Total # W↔W	-	$Ib/(mE)$

Table III: Communication complexities for both MD-GAN and FL-GAN. C and W stand for the central server and the workers, respectively.

κb (with $\kappa \leq N$), but only two batches are sent per worker. The total communication from the server is thus $2bdN$ (*i.e.*, $2bd$ per worker).

- Worker to server communications: after computing the generator errors on $X_n^{(g)}$, all workers send their error term F_n to the server. The size of error term is bd per worker, because solely one float is required for each feature of the data.
- Worker to worker communications: after E local epochs, each discriminator parameters are swapped. Each worker sends a message of size $|\boldsymbol{\theta}_n|$, and receive a message of the same size (as we assume for simplicity that discriminator models on workers have the same architecture).

Communication complexities are summarized in Table III, for both MD-GAN and FL-GAN. Table IV instantiates those complexities with the actual quantities of data measured for the experiment on the CIFAR10 dataset. The first observation is that MD-GAN requires server to workers communication at every iteration, while FL-GAN performs mE/b iterations in between two communications. Note that the size of workers-server communications depends on the GAN parameters ($\boldsymbol{\theta}$ and $\boldsymbol{w}$) for FL-GAN, whereas it depends on the size of data objects and on the batch size in MD-GAN. It is particularly interesting to choose a small batch sizes, especially since it is shown by Gupta et al. [31] that in order to hope for good performances in the parallel learning of a model (as discriminators in MD-GAN), the batch size should be inversely proportional to the number of workers N. When the size of data is around the number of parameters of the GAN (such as in image applications), the MD-GAN communications may be expensive. For example, GoogLeNet [32] analyzes images of 224×224 pixels in RGB ($150,528$ values per data) with less than 6.8 millions of parameters.

We plotted on Figure 2 an analysis of the maximum ingress traffic (x-axis) of the FL-GAN and MD-GAN schemes, for a single iteration, and depending on chosen batch size (y-axis).

Communication type	FL-GAN $b=10$	FL-GAN $b=100$	MD-GAN $b=10$	MD-GAN $b=100$
C→W (C)	175 MB	175 MB	2.30 MB	23.0 MB
C→W (W)	17.5 MB	17.5 MB	0.23 MB	2.30 MB
W→C (W)	17.5 MB	17.5 MB	0.23 MB	2.30 MB
W→C (C)	175 MB	175 MB	2.30 MB	23.0 MB
Total # C↔W	100	1,000	50,000	50,000
W→W (W)	-	-	6.34 MB	6.34 MB
Total # W↔W	-	-	100	1,000

Table IV: Illustration of communication costs for both MD-GAN and FL-GAN, in the CIFAR10 experiment with 10 workers.

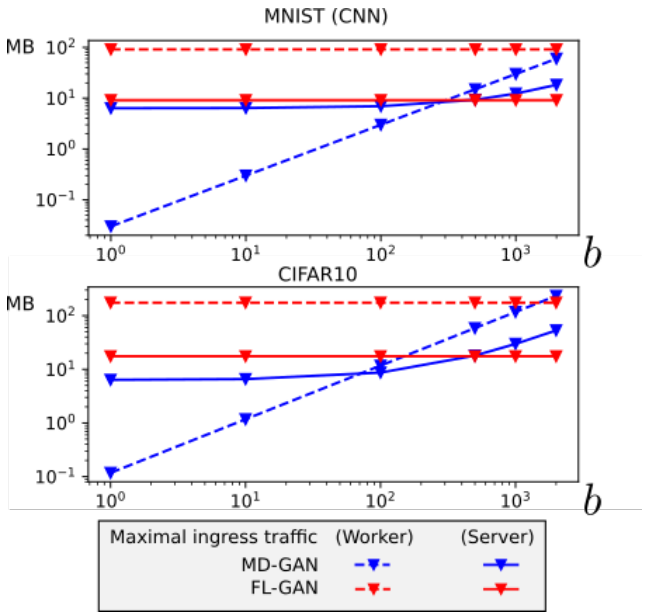

Figure 2: Maximal ingress traffic, per communication, for two types of GANs (for both MD-GAN and FL-GAN).

This corresponds for FL-GAN to a worker-server communication, and for MD-GAN for both worker-server and worker-worker communications during an iteration. Plain lines depict the ingress traffic at workers, while dotted lines the traffic at the server; these quantities can help to dimension the network capabilities required for the learning process to take place. Note the log-scale on both axis.

As expected the FL-GAN traffic is constant, because the communications depends only on the model sizes that constitute the GAN; it indicates a target upper bound for the efficiency of MD-GAN. MD-GAN lines crossing FL-GAN is indicating more incurring traffic with increasing batch sizes. A global observation is that MD-GAN is competitive for smaller batch sizes, yet in the order of hundreds of images (here of less than around $b = 550$ for MNIST and $b = 400$ for CIFAR10).

2) Computation complexity: The goal of MD-GAN is to remove the generator tasks from workers by having a single one at the server. During the traning of MD-GAN, the traffic between workers and the server is reasonable (Table III). The complexity gain on workers in term of memory and computation depends on the architecture of $\mathcal{D}$; it is generally

half of the total complexity because $\mathcal{G}$ and $\mathcal{D}$ are often similar. The consequence of this single generator-based algorithm is more frequent interactions between workers and the server, and the creation of a worker-to-worker traffic. The overall operation complexities are summarized and compared in Table II, for both MD-GAN and FL-GAN; the Table indicates a workload of half the one of FL-GAN on workers.

V. Experimental evaluation

We now analyze empirically the convergence of MD-GAN and of competing approaches.

A. Experimental setup

Our experiments are using the Keras framework with the Tensorflow backend. We emulated workers and the server on GPU-based servers equipped of two Intel Xeon Gold 6132 processor, 260 GB of RAM and four NVIDIA Tesla M60 GPUs or four NVIDIA Tesla P100 GPUs. This setup allows for a training of GANs that is identical to a real distributed deployement, as computation order of interactions for Algorithm IV are preserved. This choice for emulation is thus oriented towards a tighter control for the environment of competing approaches, to report more precise head to head result comparisons; raw timing performances of learning tasks are in this context inaccessible and are left to future work.

a) Datasets: We experiment competing approaches on two classic datasets for deep learning: MNIST [33] and CIFAR10 [34]. MNIST is composed of a training dataset of $60,000$ grayscale images of 28×28 pixels representing handwritten digits and another test dataset of $10,000$ images. Theses two datasets are composed respectively of $6,000$ and $1,000$ images for each digits. CIFAR10 is composed of a training set $50,000$ RGB images of 32×32 pixels representing the followings 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. CIFAR10 has a test dataset of $10,000$ images. We later validate our conclusions on the CelebA dataset [35], which is composed by 200K images of celebrities (128×128 pixels).

b) GAN architectures: In the experiments, we train a classical type of GAN named ACGAN [19]. We experiment with three different architectures for $\mathcal{G}$ and $\mathcal{D}$: a multi-layer based architecture (MLP), a convolutional neural network based architecture (CNN) for MNIST and a CNN-based architecture for CIFAR10. Their characteristics are:

- In the MLP-based architecture for MNIST, $\mathcal{G}$ and $\mathcal{D}$ are composed of three fully-connected layers each. $\mathcal{G}$ layers contain respectively 512, 512 and 784 neurons, and $\mathcal{D}$ layers contain 512, 512 and 11 neurons. The total number of parameters is $716,560$ for $\mathcal{G}$ and $670,219$ for $\mathcal{D}$.
- In the CNN-based architecture for MNIST, $\mathcal{G}$ is composed of one full-connected layer of $6,272$ neurons and two transposed convolutional layers of respectively 32 and 1 kernels of size 5×5. $\mathcal{D}$ is composed of six convolutional layers of respectively 16, 32, 64, 128, 256 and 512 kernels of size 3×3, a mini-batch discriminator layer [20] and one full-connected layer of 11 neurons. The total number of parameters is $628,058$ for $\mathcal{G}$ and $286,048$ for $\mathcal{D}$.
- In the CNN-based architecture for CIFAR10, $\mathcal{G}$ is composed of one full-connected layer of $6,144$ neurons and three transposed convolutional layers of respectively 192, 96, and 3 kernels of size 5×5. $\mathcal{D}$ is composed of six convolutional layers of respectively 16, 32, 64, 128, 256 and 512 kernels of size 3×3, a mini-batch discriminator layer and one full-connected layer of 11 neurons. The total number of parameters is $628,110$ for $\mathcal{G}$ and $100,203$ for $\mathcal{D}$.

c) Metrics: Evaluating generative models such as GANs is a difficult task. Ideally, it requires human judgment to assess the quality of the generated data. Fortunately, in the domain of GANs, interesting methods are proposed to simulate this human judgment. The main one is named the *Inception Score* (we denote it by IS), and has been proposed by Salimans *et al.* [20], and shown to be correlated to human judgment. The IS consists to apply a pre-trained Inception classifier over the generated data. The Inception Score evaluates the confidence on the generated data classification (*i.e.*, generated data are well recognized by the Inception network), and on the diversity of the output (*i.e.*, generated data are not all the same). To evaluate the competitors on MNIST, we use the MNIST score (we name it MS), similar to the Inception score, but using a classifier adapted to the MNIST data instead of the Inception network. Heusel *et al.* propose a second metric named the Fréchet Inception Distance (FID) in [36]. The FID measures a distance between the distribution of generated data $P_{\mathcal{G}}$ and real data P_{data}. It applies the Inception network on a sample of generated data and another sample of real data and supposes that their outputs are Gaussian distributions. The FID computes the Fréchet Distance between the Gaussian distribution obtained using generated data and the Gaussian distribution obtained using real data. As for the Inception distance, we use a classifier more adapted to compute the FID on the MNIST dataset. We use the implementation of the MS and FID available in Tensorflow[3].

d) Configurations of MD-GAN and competing approaches: To compare MD-GAN to classical GANs, we train the same GAN architecture on a standalone server (it thus has access to the whole dataset $\mathcal{B}$). We name this baseline *standalone-GAN* and parametrize it with two batch sizes $b = 10$ and $b = 100$.

We run FL-GAN with parameters $E = 1$ and $b = 10$ or $b = 100$; this parameter setting comes from the fact that $E = 1$ and $b = 10$ is one of the best configuration regarding computation complexity on MNIST, and because $b = 50$ is the best one for performance per iteration [15] (having $b = 100$ thus allows for a fair comparison for both FL-GAN and MD-GAN). MD-GAN is run with also $E = 1$; *i.e.*, for FL-GAN and MD-GAN, respective actions are taken after the whole dataset has been processed once.

[3]Code available at https://github.com/tensorflow/models/blob/master/research/gan/mnist/util.py.

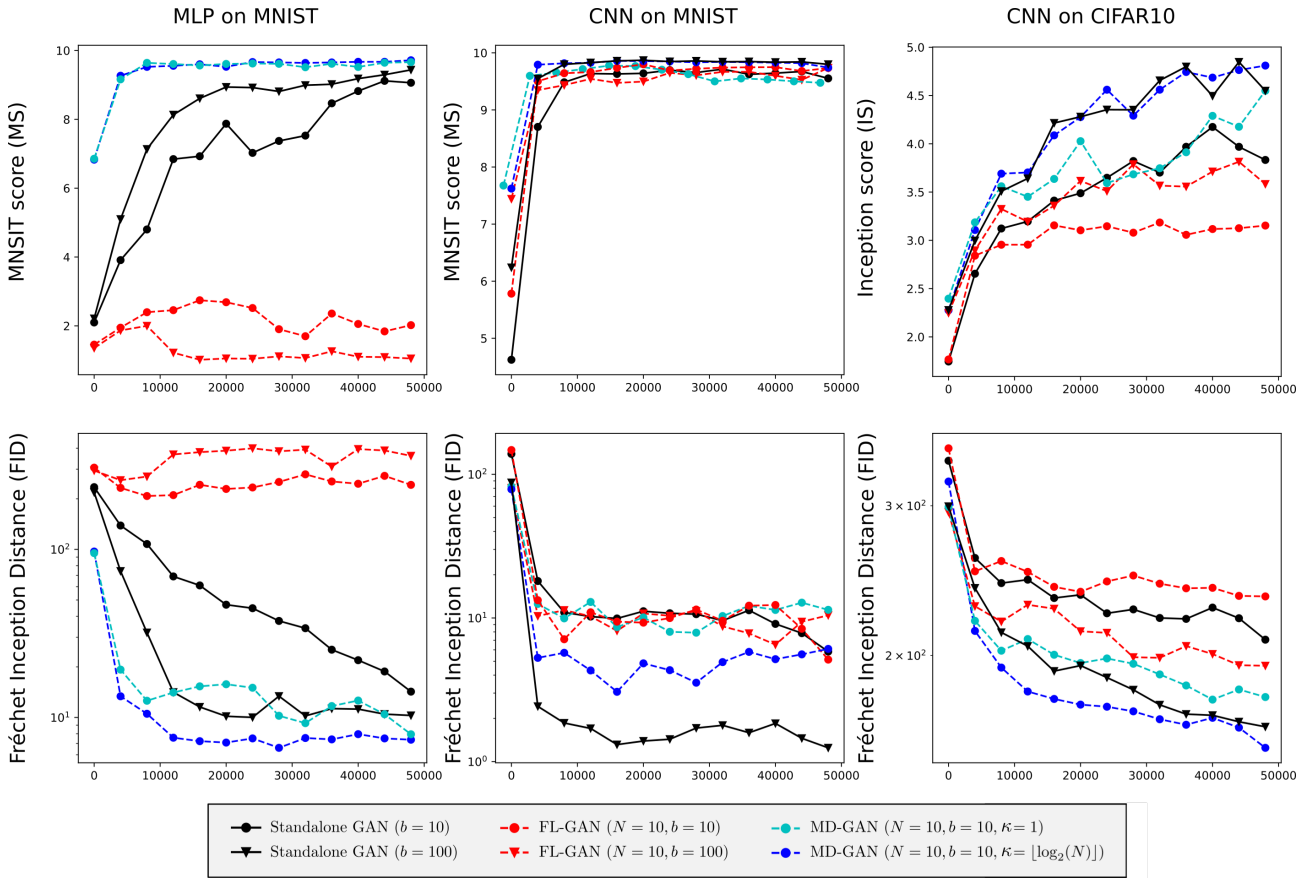

Figure 3: MNIST score / Inception score (higher is better) and Fréchet Inception Distance (lower is better) for the three competing approaches, with regards to the number of iterations (x-axis).

For MD-GAN and FL-GAN, the training dataset is split equally over workers (images are sampled $i.i.d$). We run two configurations of MD-GAN, one with $\kappa = 1$ and another with $\kappa = \lfloor \log(N) \rfloor$, in order to evaluate the impact of the data diversify sent to workers. Finally, in FL-GAN, GANs over workers perform learning iterations (such as in the standalone case) during 1 epoch, *i.e.*, until $\mathcal{D}_n$ processes all local data $\mathcal{B}_n$.

We experimented with a number of workers $N \in [1, 10, 25, 50]$; geo-distributed approaches such as Gaia [8] or [9] also operate at this scale (where 8 nodes [9] and 22 nodes [8] at maximum are leveraged). All experiments are performed with $I = 50,000$, *i.e.*, the generator (or the N generators in FL-GAN) are updated $50,000$ times during a generator learning step. We compute the FID, MS and IS scores every $1,000$ iterations using a sample of 500 generated data. The FID is computed using a batch of the same size from the test dataset. In FL-GAN, the scores are computed using the generator on the central server.

B. Experiment results

We report the scores of all competitors, with regards to the iterations, on the Figure 3. The resulting curves are smoothed for readability.

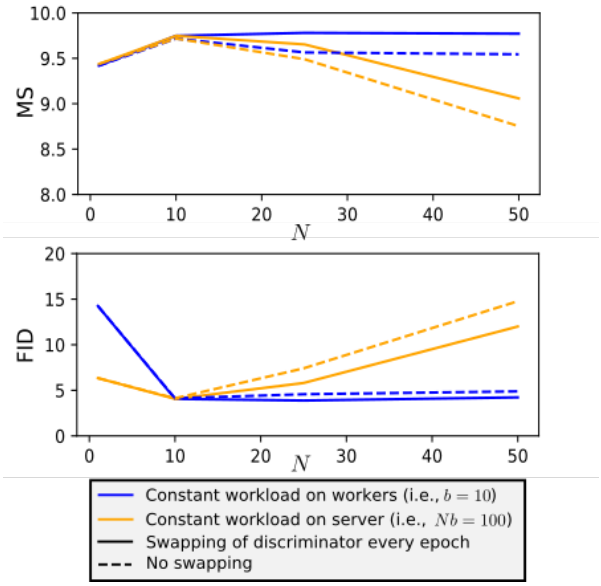

Figure 4: MNIST score and Fréchet Inception Distance with regards to the varying number of workers (x-axis) for MD-GAN using the MLP model. Experiments include the disabling of the swapping processing for comparison purposes.

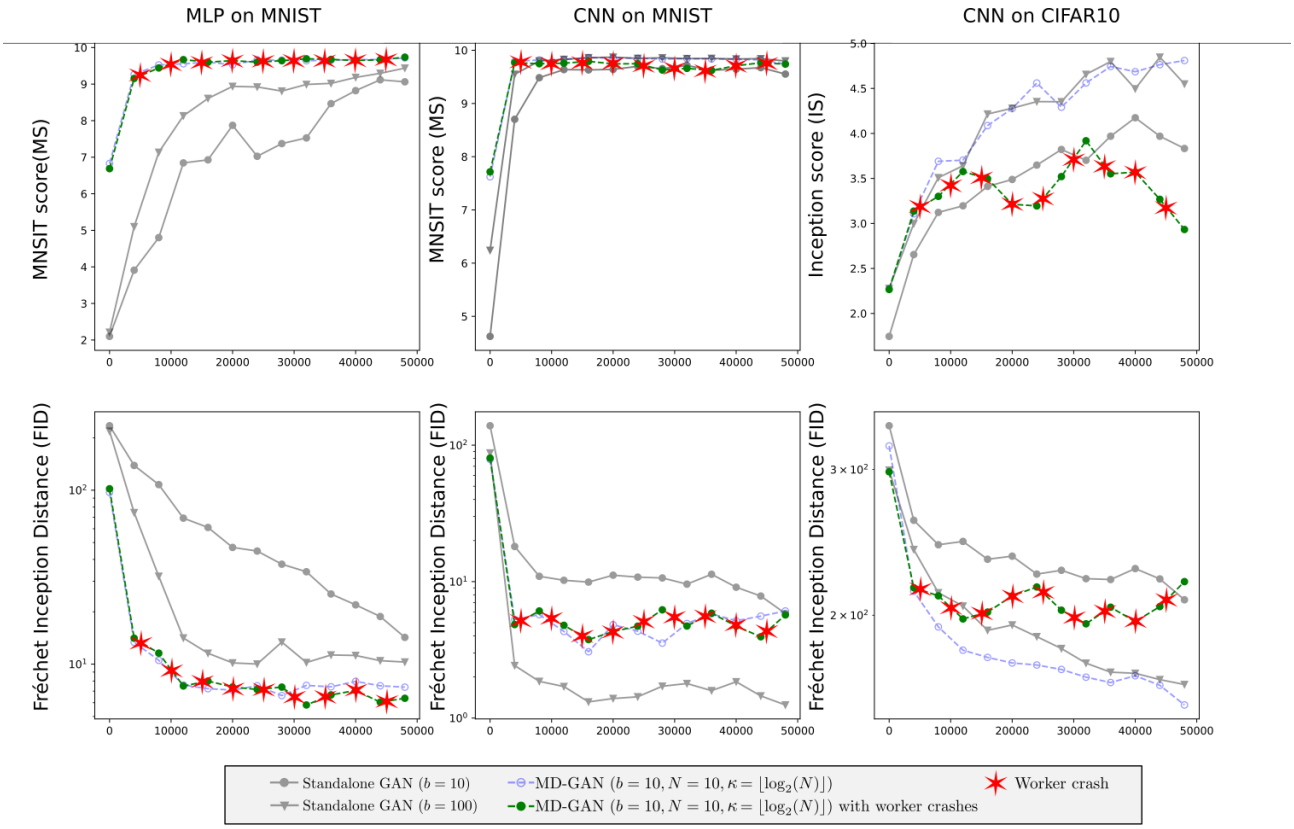

Figure 5: MNIST score or Inception score and Fréchet Inception Distance over the number of iterations for MD-GAN with crash faults, compared to MD-GAN without any crash and to a standalone GAN.

1) Competitor scores: The standalone GAN obtains better results with $b = 100$, rather than with $b = 10$. It is because the GAN sees more samples (real and generated data) per iteration when b increases. When $b = 10$ for MD-GAN, the total number of real data seen in all $\mathcal{B}_n$ is 100 with $N = 10$. This explains why MD-GAN obtains very similar scores than standalone GAN with $b = 100$ (except with the CNN on MNIST). We note that, as highlighted in discussion in Section IV-B4, the hyper-parameter κ has a significant impact on the learning process. The more the data diversity sent by the server to workers, the higher the generator scores.

For the experiments on MLP, FL-GAN does not converge, whereas MD-GAN has better scores (FID and MS) than the standalone competitor. We propose a multi-discriminator vs one generator game; some recent works [10] have shown that some central strategies based on one generator and multiple discriminators, or a *mixture* of generators and one discriminator [11], can as well exceed the performances of a standalone GAN. In the CNN experiments on MNIST, the FID and MS scores obtained by MD-GAN and FL-GAN are close to equivalent. In the CNN experiments on CIFAR10, MD-GAN obtains better IS and MS than FL-GAN over this more complex learning task.

These three experiments show that MD-GAN exploits the advantage of having a single generator to train, that faces multiple discriminators.

2) Scalability and the impact of worker to worker communications: We present on Figure 4 the evolution of the final accuracy score for MD-GAN (after $20,000$ iterations), as a function of the number of workers using the MLP model. Because the dataset is split over workers, increasing the number of participants reduces the size of local datasets ($|\mathcal{B}_n| = |\mathcal{B}|/N$).

Two variants of MD-GAN are executed. The first one is the discussed MD-GAN algorithm, and the second one depicts on the dotted curves MD-GAN where no swapping between workers occurs (*i.e.*, with respectively $E = 1$, and $E = \infty$). The blue curves present the MD-GAN scores when the workload on workers (*i.e.*, the number of images to process) remains constant, while the orange curves present a constant workload of the central node. We note that Figure 4 also illustrates a varying size of mini-batches b used by workers on the curve with a constant workload on server: the larger N is, the lower b is in consequence, to maintain the same workload on the server.

We note that interesting phenomenons appear at scale after $N = 10$; for lower values of N the workers appear to have enough data locally to reach satisfying scores.

The first observation is that considering a constant workload on workers leads to better results. This yet comes at the price of a higher cost on the server (cf Table II and III).

The swapping process between workers leads to better

results. We yet observe that, despite the better result in MS, the FID score improvement using swapping is marginal in the case of the constant workload on server setting. This indicates that data available locally to workers is enough, and that their is a marginal gain to await from the diversity brought by swapping discriminators.

3) Fault tolerance of MD-GAN facing worker crashes: In order to assess the tolerance of a MD-GAN learning task facing worker fail-stop crashes (workers' data also disappear from the system when the crash occurs), we conduct the following experiment, presented in Figure 5. We operate in the same scenario than for experiments in Figure 3, and for the best performing MD-GAN setup (with $\kappa = \lfloor \log(N) \rfloor$), but this time we trigger a worker to crash every I/N iterations (appearing as the curve in green). Consequence is that at $I = 50,000$, all workers have crashed. For comparison with a baseline, standalone GAN (*i.e.*, single server GAN learning) are reploted for two batch sizes ($b \in [10, 100]$), and so is the non crashing run on the blue curve with same parametrization.

First observation is that this crash pattern has a no significant impact on the result performance for the MNIST dataset, for both MS and FD metrics. The MLP architecture even exhibits the smallest FD at the end of the experiment. This highlights that for this dataset, the MD-GAN architecture manages to learn fast enough so that crashes, and then the removal of dataset shares, are not a problem performance-wise.

Both metrics are affected in the case of the CIFAR10 dataset: we observe a divergence due to crashes, and it happens early in the learning phase (around $I = 5,000$, corresponding to the first crashed worker). This experiment shows the sensitivity of the learning to early failures, because GANs did not have enough time to accurately approximate the distribution of the data, and then misses the lost data shares for reaching a competitive score. Scores are yet comparable to the standalone baseline up to 8 crashed workers.

We nevertheless note that in the geo-distributed learning frameworks [8], [9] that our work is aiming to support, the crashes of several workers will undoubtedly trigger repair mechanisms, in order to cope with diverging learning tendencies.

4) Validation on a larger dataset: In this experiment, we validate the convergence of MD-GAN, and its interest with regards to the standalone and FL-GAN approaches. The goal is to train a GAN over the CelebA dataset [35]. We use 10K of the 200K images in CelebA as the test dataset, while the remaining images are distributed equally (*i.i.d.*) over the $N = \{1, 5\}$ workers. The GAN architecture is a variant of the one used for the CIFAR10 dataset: $\mathcal{G}$ is composed of one fully-connected layer of $16,384$ neurons and two transposed convolutional layers of respectively 128 and 3 kernels of size 5×5; $\mathcal{D}$ is composed of six convolutional layers of respectively 16, 32, 64, 128, 256 and 512 kernels of size 3×3, and one fully-connected layer of one neuron. The batch size for the standalone GAN and FL-GAN is $b = 200$ whereas the batch size of MD-GAN is $b = 40$ (corresponding to 200 images processed to compute one generator update). In this experiment, we use two different settings for the Adam optimizer, leading to better results for each competitor. The standalone GAN and FL-GAN use a learning rate of $\alpha = 0.003$ for $\mathcal{G}$ (resp. $\alpha = 0.002$ for $\mathcal{D}$), and hyperparameters $\beta_1 = 0.5$, $\beta_2 = 0.999$ for both, whereas MD-GAN uses a learning rate of $\alpha = 0.001$ for $\mathcal{G}$ (resp. $\alpha = 0.004$ for $\mathcal{D}$), and hyperparameters $\beta_1 = 0.0$, $\beta_2 = 0.9$ for both. The resulting FID and Inception scores during the $30,000$ iterations we considered are reported in Figure 6. We observe that all IS scores are comparable (MD-GAN is slightly above); yet regarding the FID, MD-GAN (as well as FL-GAN) is distanced by the standalone approach (as this is the case for the CNN experiment on MNIST).

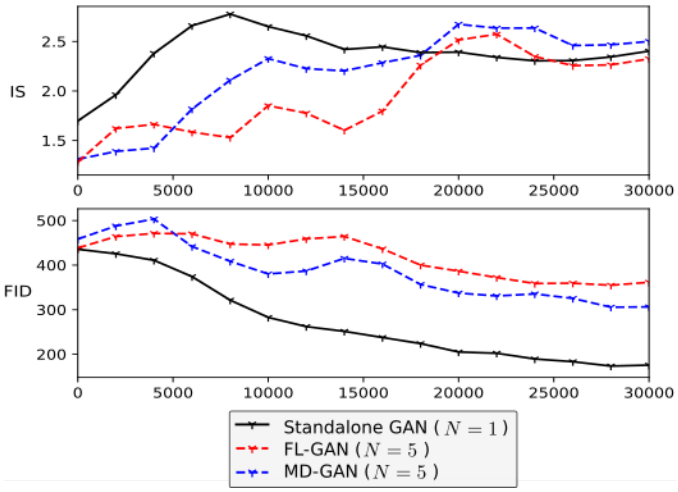

Figure 6: Inception scores and Fréchet Inception Distance of the three competitors, on the CelebA dataset.

VI. Related Work: Distributing Deep Learning

Distributing the learning of deep neural networks over multiple machines is generally performed with the Parameter Server model proposed by J. Dean et al in [22]. This model was adapted in different works [31], [14], [37]. The first interest is to speed up the learning in large data-centers [12], [38], [39]. This parameter server model was used for privacy reasons in [40]. The federated learning is a most accomplished method using the parameter server model with auxiliary workers to reduce communications [27] or increase the privacy [41].

We experimented in a position paper [24] the distribution of the generator function. In this fully decentralized setup where compute nodes exchange their generators and discriminators in a gossip fashion (there are n couples of generator and discriminators, one per worker), the experiment results are favorable to federated learning. We then propose MD-GAN as a solution for a performance gain over federated learning.

Finally, a recent work [42] proposes to multiply the number of discriminators and generators in a datacenter location: the authors propose to train several couples of GAN in parallel and to swap generators and discriminators every fix amount

of iterations. Durugkar et al. [10] propose a centralized multi-discriminators architecture to improve the discriminator judgment on generated data. In the same way, Hoang et al. [11] study a centralized multi-generator architecture is proposed to improve the generator capacities and to reduce the so called *mode collapse problem* [17]. The works [43] and [44] improve the mixture of generative adversarial models. Wang et al. [43] use ensemble of GANs trained separately organized as a cascade to build an aggregated model. In the work of Tolstikhin et al. [44], GANs are trained sequentially using boosting strategies to incrementally improve the performance of the final model. Note that all these works are proposed to improve GAN convergence, but not to distribute the learning (discriminators have access to the whole dataset). Our contribution is a method leveraving multiple adversaries in a distributed setup, and taking the network constraints into account.

VII. Perspectives and conclusion

Before we conclude, we highlight the salient questions on the way to a widespread distribution of GANs.

1) Asynchronous setting: Instead waiting all F every global iteration, the server may compute a gradient Δw and apply it each time it receives a single F_n. Fresh batches of data can be generated frequently, so that they can be sent to idle workers. All workers can operate without global synchronization, contrarily to federated learning methods as FL-GAN. In this setting, the waiting times of both workers and the server are reduced drastically. However, because of asynchronous updates, there is no guarantee that the parameters $\boldsymbol{w}$ of a worker n at time t (used to generate $X_n^{(g)}$) are the same at time $t + \Delta t$ when it sends its F_n to the server.

In the parameter server model, asynchrony implies inconsistent updates by workers. In practice, the training task nevertheless works well if the learning rate is adapted in consequence [14], [31], [13].

2) The central server communication bottleneck: The parameter server framework, despite its simplicity, has the obvious drawback of creating a communication bottleneck towards the central server. This has been quantified by several works [12], [13], and solutions for traffic reduction between workers and the server have been proposed. Methods such as Adacomp [13] propose to communicate updates based on gradient staleness, which constitutes a form of data compression.

In the context of GANs, those methods may be applied on generated data before they are sent to workers, and to the error feedback messages sent by workers to the server. In particular, concerning images, there are many techniques from their compression (with or without loss of information, see *e.g.*, [45]).

A fine grained combination of techniques for gradient and data object compression would make the parameter server framework more sustainable for GANs and increasingly larger datasets to learn on. A second direction might be to mix the federated learning approach with ensemble of GANs training independently in cascades (as presented by Wang et al. [43]). Federated learning would act as the scheduling mechanism for the parallel ensembles; this would restrict the burden on the server to critical only communications (up-to-date model hosting and dispatching), while most of the training occurs on edge workers, hierarchically.

3) Adversaries in generative adversarial networks: The current deployment setup of GANs in the literature is assuming an adversary-free environment. In fact, the question of the capacity of basic deep learning mechanisms to embed byzantine fault tolerance has just been recently proposed for distributed gradient descent [46]. In addition to the gradient updates in GANs, and more specifically, the learning process is most likely prone to workers having their discriminator lie to the server's generator (by sending erroneous or manipulated feedback). The global convergence, and then the final performance of the learning task will be affected in an unknown proportion. This adversarial setup, and more generally better fault tolerance, are a crucial aspect for future applications in the domain.

4) Scaling the number of workers: We experimented MD-GAN over up to 50 parallel workers. The current scale at which parallel deep learning is operating is in the order to tens (*e.g.*, in Gaia [8] or in [9]) to few hundreds of workers (experiments in TensorFlow [47] for instance reach 256 workers maximum). It is still not well understood what is the bottleneck for reaching larger scales: is the dataset size imposing the scale? Or is this the conflicting asynchronous updates [14] from workers to the server limiting the benefit of scale after a certain threshold? We note that federated learning can be used on a large number of workers (*e.g.*, $2,000$ in some works [27]) by using only a random subset of the available devices at every round. MD-GAN can be adapted in a similar way, with fewer discriminators than workers: because discriminator models are swapped during the learning process, the whole distributed dataset could be leveraged.

Those general questions for deep learning are also applying to the learning of GANs, as they are themselves constituted by couples of deep neural networks. The unknown spot comes from the specificity of GANs, because of the coupling of generators and discriminators; that coupling will most likely play an additional major role in the future algorithms that will be dedicated to push the scalability of GANs to a new standard.

This paper has presented generative adversarial networks in the novel context of parallel computation and of learning over distributed datasets; MD-GAN aims at being leveraged by geo-distributed or edge-device deep learning setups. We have presented an adaptation of federated learning to the problem of distributing GANs, and shown that it is possible to propose an algorithm (MD-GAN) that removes half the computation complexity from workers by using a discriminator swapping technique, while still achieving better results on the two reviewed datasets. GANs are computationaly and communication intensive, specially in the considered data-distributed setup; we believe this work brought a first viable

solution to that domain. We hope that raised perspectives will trigger interesting future works for the system and algorithmic support of the nascent field of generative adversarial networks.

REFERENCES

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *ArXiv e-prints*, Jun. 2014.

[2] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative Adversarial Text to Image Synthesis," *ArXiv e-prints*, May 2016.

[3] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating Videos with Scene Dynamics," *ArXiv e-prints*, Sep. 2016.

[4] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," *CVPR*, 2017.

[5] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez, "Invertible Conditional GANs for image editing," *ArXiv e-prints*, Nov. 2016.

[6] M. Chidambaram and Y. Qi, "Style transfer generative adversarial networks: Learning to play chess differently," *CoRR*, vol. abs/1702.06762, 2017.

[7] E. J. Hyunsun Choi, "Generative ensembles for robust anomaly detection," *CoRR*, vol. abs/1810.01392v1, 2018.

[8] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *NSDI*, 2017.

[9] I. Cano, M. Weimer, D. Mahajan, C. Curino, and G. M. Fumarola, "Towards geo-distributed machine learning," *CoRR*, vol. abs/1603.09035, 2016.

[10] I. Durugkar, I. Gemp, and S. Mahadevan, "Generative Multi-Adversarial Networks," *ICLR*, Nov. 2016.

[11] Q. Hoang, T. Dinh Nguyen, T. Le, and D. Phung, "Multi-Generator Generative Adversarial Nets," *ArXiv e-prints*, Aug. 2017.

[12] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014.

[13] C. Hardy, E. Le Merrer, and B. Sericola, "Distributed deep learning on edge-devices: Feasibility via adaptive compression," in *NCA*, 2017.

[14] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," in *IJCAI*, 2016.

[15] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[17] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," *ArXiv e-prints*, Jan. 2017.

[18] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *ArXiv e-prints*, Jan. 2017.

[19] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 2642–2651.

[20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *ArXiv e-prints*, Jun. 2016.

[21] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *SOSP*, 2007.

[24] C. Hardy, E. Le Merrer, and B. Sericola, "Gossiping GANs," in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning: DIDL*, 2018.

[22] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *NIPS*, 2012.

[23] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," *ArXiv e-prints*, Nov. 2016.

[25] Y. Wang, L. Zhang, and J. van de Weijer, "Ensembles of generative adversarial networks," in *NIPS 2016 Workshop on Adversarial Training*, 2016.

[26] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. SIMON-GABRIEL, and B. Schölkopf, "Adagan: Boosting generative models," in *NIPS*, 2017.

[27] J. Konečný, H. Brendan McMahan, F. X. Yu, P. Richtárik, A. Theertha Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," *CoRR*, vol. abs/1610.05492, Oct. 2016.

[28] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011.

[29] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *NIPS*, 2015.

[30] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting Distributed Synchronous SGD," *arXiv e-prints*, p. arXiv:1604.00981, Apr. 2016.

[31] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *ICDM*, 2016.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[33] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist, 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[35] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *ICCV*, 2015.

[36] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," *ArXiv e-prints*, Jun. 2017.

[37] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *NIPS*, 2015.

[38] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *OSDI*, 2014.

[39] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2016.

[40] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS*, 2015.

[41] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy preserving machine learning," Cryptology ePrint Archive, Report 2017/281, 2017.

[42] D. Jiwoong Im, H. Ma, C. Dongjoo Kim, and G. Taylor, "Generative Adversarial Parallelization," *ArXiv e-prints*, Dec. 2016.

[43] Y. Wang, L. Zhang, and J. van de Weijer, "Ensembles of Generative Adversarial Networks," *arXiv e-prints*, p. arXiv:1612.00991, Dec. 2016.

[44] I. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf, "AdaGAN: Boosting Generative Models," *arXiv e-prints*, p. arXiv:1701.02386, Jan. 2017.

[45] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug 2000.

[46] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017.

[47] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.